Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №2 по курсу «Моделирование»

Тема Метод Рунге-Кутта 4-го порядка при решении системы ОДУ

Студент Кононенко С.С.

Группа ИУ7-63Б

Оценка (баллы) ____________________

Преподаватель Градов В.М.

Москва — 2021 г.

Тема работы

Программно-алгоритмическая реализация метода Рунге-Кутта 4-го порядка точности при решении системы ОДУ в задаче Коши.

Цель работы

Получение навыков разработки алгоритмов решения задачи Коши при реализации моделей, построенных на системе ОДУ, с использованием метода Рунге-Кутта 4-го порядка точности.

Теоретические сведения

Опишем колебательный контур с помощью системы уравнений:

$$\begin{cases} L_k \frac{dI}{dt} + (R_k + R_p(I)) \cdot I - U_C = 0 \\ \frac{dU_c}{dt} = -\frac{I}{C_k} \end{cases}$$

Значение $R_p(I)$ можно вычислить по формуле:

$$Rp = \frac{l_e}{2\pi \cdot \int_0^R \sigma(T(r))rdr} = \frac{l_e}{2\pi R^2 \cdot \int_0^1 \sigma(T(z))dz}$$

т. к. $z = r/R$.

Значение $T(z)$ вычисляется по формуле:

$$T(z) = T_0 + (T_w - T_0) \cdot Z^m$$

Заданы начальные параметры:

$R = 0.35$ см (Радиус трубки)

$l_e = 12$ см (Расстояние между электродами лампы)

$L_k = 187 * 10^{-6}$ Гн (Индуктивность)

$C_k = 268 * 10^{-6}$ Ф (Емкость конденсатора)

$R_k = 0.25$ Ом (Сопротивление)

$U_{c0} = 1400$ В (Напряжение на конденсаторе в начальный момент времени)

$I_0 = 0.3$ А (Сила тока в цепи в начальный момент времени t = 0)

$T_w = 2000$ K

Метод Рунге-Кутта четвертого порядка точности

Имеем систему уравнений вида:

$$\begin{cases} u'(x) = f(x, u(x)) \\ u(\xi) = \eta \end{cases}$$

Тогда:

$y_{n+1} = y_n + \frac{k_1+2k_2+2k_3+k_4}{6}$

$k_1 = h_n f(x_n, y_n)$

$k_2 = h_n f(x_n + \frac{h_n}{2}, y_n + \frac{k_1}{2})$

$k_3 = h_n f(x_n + \frac{h_n}{2}, y_n + \frac{k_2}{2})$

$k_4 = h_n f(x_n + h_n, y_n + k_3)$

Рассмотрим обобщение формулы на случай двух переменных. Пусть дана система:

$$\begin{cases} u'(x) = f(x, u, v) \\ v'(x) = \varphi(x, u, v) \\ v(\xi) = v_0 \\ u(\xi) = u_0 \end{cases}$$

Тогда:

$y_{n+1} = y_n + \frac{k_1+2k_2+2k_3+k_4}{6}$

$z_{n+1} = z_n + \frac{q_1+2q_2+2q_3+q_4}{6}$

$$k_1 = h_n f(x_n, y_n, z_n)$$
$$k_2 = h_n f(x_n + \frac{h_n}{2}, y_n + \frac{k_1}{2}, z_n + \frac{q_1}{2})$$
$$k_3 = h_n f(x_n + \frac{h_n}{2}, y_n + \frac{k_2}{2}, z_n + \frac{q_2}{2})$$
$$k_4 = h_n f(x_n + h_n, y_n + k_3, z_n + q_3)$$
$$q_1 = h_n \varphi(x_n, y_n, z_n)$$
$$q_2 = h_n \varphi(x_n + \frac{h_n}{2}, y_n + \frac{k_1}{2}, z_n + \frac{q_1}{2})$$
$$q_3 = h_n \varphi(x_n + \frac{h_n}{2}, y_n + \frac{k_2}{2}, z_n + \frac{q_2}{2})$$
$$q_4 = h_n \varphi(x_n + h_n, y_n + k_3, z_n + q_3)$$

Исходный код алгоритмов

В листинге 1 представлена реализация алгоритма решения задачи. В листингах 2 – 6 приведены вспомогательные функции и главная программа.

Листинг 1 – Реализация алгоритма решения задачи

```
package circuit

import (
    "fmt"
    "math"

    "gonum.org/v1/gonum/integrate/quad"
    "gonum.org/v1/gonum/interp"
)

// GetT0 is used to find T0 parameter
func GetT0(I float64) float64 {
    return interpolate(I, CurTbl.GetColumn(0), CurTbl.GetColumn(1))
}

// GetM is used to find m parameter
func GetM(I float64) float64 {
    return interpolate(I, CurTbl.GetColumn(0), CurTbl.GetColumn(2))
}

// GetRp is used to find Rp parameter
func GetRp(I, T0, m float64) float64 {
    f := func(x float64) float64 {
        return getSigma(getT(x, T0, m)) * x
    }
    val := quad.Fixed(f, 0, 1, 30, nil, 0)
```

```go

    return Params.Le / (2 * math.Pi * Params.R * Params.R * val)
}

// GetRungeKutta is used to find parameters with Runge-Kutta method
func GetRungeKutta(x, y, z, h, Rp float64) (float64, float64) {
    cfsArr := make([]RCoeffs64, Order)

    for i := 0; i < Order; i++ {
        v := i
        if i == 0 {
            v = Order - 1
        }
        _, yAdd, zAdd := getCurAdd(h, cfsArr[v], i, Order)
        cfsArr[i] = RCoeffs64{h * getF(y+yAdd, z+zAdd, Rp), h * getPhi(y+yAdd)}
    }

    return getNextMembs(y, z, cfsArr)
}

func getT(z, T0, m float64) float64 {
    return (Params.Tw-T0)*math.Pow(z, m) + T0
}

func getF(y, z, Rp float64) float64 {
    return -((Params.Rk+Rp)*y - z) / Params.Lk
}

func getPhi(y float64) float64 {
    return -y / Params.Ck
}

func getSigma(T float64) float64 {
    return interpolate(T, TmpTbl.GetColumn(0), TmpTbl.GetColumn(1))
}

func interpolate(x float64, xs, ys FArr64) float64 {
    var as interp.AkimaSpline

    err := as.Fit(xs, ys)
    if err != nil {
        fmt.Println("Failed to initialize spline")
    }

    return as.Predict(x)
}

func getCurAdd(h float64, cfs RCoeffs64, i, ord int) (float64, float64, float64) {
```

```
    if i == 0 {
        return 0, 0, 0
    }
    if i == ord-1 {
        return h, cfs.Kn, cfs.Pn
    }
    return h / 2, cfs.Kn / 2, cfs.Pn / 2
}

func getNextMembs(y, z float64, cfsArr []RCoeffs64) (float64, float64) {
    var (
        kSum float64 = 0
        pSum float64 = 0
        div float64 = float64(2*(len(cfsArr)-2) + 2)
    )

    for i := 0; i < len(cfsArr); i++ {
        if i > 0 && i < len(cfsArr)-1 {
            kSum += 2 * cfsArr[i].Kn
            pSum += 2 * cfsArr[i].Pn
        } else {
            kSum += cfsArr[i].Kn
            pSum += cfsArr[i].Pn
        }
    }

    return y + kSum/div, z + pSum/div
}
```

Листинг 2 – Реализация вспомогательных типов

```
package circuit

// FArr64 is used to represent []float64
type FArr64 []float64

// FMat64 is used to represent [][]float64
type FMat64 []FArr64

func (m FMat64) GetColumn(n int) FArr64 {
    var (
        c FArr64 = make(FArr64, 0)
        cs int = len(m)
    )

    for i := 0; i < cs; i++ {
        c = append(c, m[i][n])
    }

```

```go
    return c
}

// RCoeffs64 is used to represent Runge-Kutta coefficients
type RCoeffs64 struct {
    Kn float64
    Pn float64
}

// Circuit is used to represent circuit parameters
type Circuit struct {
    R float64
    Le float64
    Lk float64
    Ck float64
    Rk float64
    Uc0 float64
    I0 float64
    Tw float64
}
```

Листинг 3 – Константы

```go
package circuit

var (
    CurTbl = FMat64{
        FArr64{0.5, 6730, 0.50},
        FArr64{1.0, 6790, 0.55},
        FArr64{5.0, 7150, 1.7},
        FArr64{10.0, 7270, 3},
        FArr64{50.0, 8010, 11},
        FArr64{200.0, 9185, 32},
        FArr64{400.0, 10010, 40},
        FArr64{800.0, 11140, 41},
        FArr64{1200.0, 12010, 39},
    }
    TmpTbl = FMat64{
        FArr64{4000, 0.031},
        FArr64{5000, 0.27},
        FArr64{6000, 2.05},
        FArr64{7000, 6.06},
        FArr64{8000, 12},
        FArr64{9000, 19.9},
        FArr64{10000, 29.6},
        FArr64{11000, 41.1},
        FArr64{12000, 54.1},
        FArr64{13000, 67.7},
        FArr64{14000, 81.5},
```

```
    }
    Params = Circuit{0.35, 12, 187e-6, 268e-6, 0.25, 1400, 1.5, 2000}
    Order int = 4
)
```

Листинг 4 – Реализация вспомогательных функций

```
package circuit

import "math"

// Arange is used to model numpy.arange behaviour
func Arange(start, stop, step float64) []float64 {
    n := int(math.Ceil((stop - start) / step))
    rnge := make([]float64, n)
    for x := range rnge {
        rnge[x] = start + step*float64(x)
    }
    return rnge
}
```

Листинг 5 – Реализация функций отрисовки графика

```
package circuit

import (
    "fmt"
    "image/color"
    "os"

    "gonum.org/v1/plot"
    "gonum.org/v1/plot/plotter"
    "gonum.org/v1/plot/vg"
)

// DrawPlot is used to draw plot with given coordinates and meta info
func DrawPlot(xs, ys FArr64, title, xl, yl, file string) {
    p := plot.New()

    p.Title.Text = title
    p.X.Label.Text = xl
    p.Y.Label.Text = yl
    p.Add(plotter.NewGrid())

    dots := convertDots(xs, ys)

    l, err := plotter.NewLine(dots)
    if err != nil {
        fmt.Println("Error:", err)
```

```
        os.Exit(1)
    }
    l.LineStyle.Width = vg.Points(1)
    l.LineStyle.Color = color.RGBA{B: 255, A: 255}

    p.Add(l)

    if err := p.Save(10*vg.Inch, 4*vg.Inch, file); err != nil {
        panic(err)
    }
}

func convertDots(xs, ys FArr64) plotter.XYs {
    var conv plotter.XYs

    for i := 0; i < len(xs); i++ {
        d := plotter.XY{
            X: xs[i],
            Y: ys[i],
        }
        conv = append(conv, d)
    }

    return conv
}
```

Листинг 6 – Главная программа

```
package main

import (
    "fmt"
    "lab_02/circuit"
)

func main() {
    var (
        I = circuit.Params.I0
        Uc = circuit.Params.Uc0
        h = 1e-6
        IRes circuit.FArr64
        RpRes circuit.FArr64
        UcRes circuit.FArr64
        T0Res circuit.FArr64
        IRpRes circuit.FArr64
        tRes circuit.FArr64
    )

    for _, t := range circuit.Arange(0, 0.0008, h) {
```

```
        T0 := circuit.GetT0(I)
        Rp := circuit.GetRp(I, T0, circuit.GetM(I))
        I, Uc = circuit.GetRungeKutta(t, I, Uc, h, Rp)

        if t > h {
            tRes = append(tRes, t)
            IRes = append(IRes, I)
            RpRes = append(RpRes, Rp)
            UcRes = append(UcRes, Uc)
            T0Res = append(T0Res, T0)
            IRpRes = append(IRpRes, I*Rp)
        }

        fmt.Printf("-- DEBUG -- Rp: %v -- I: %v -- Uc -- %v T0: %v -- Rk: %v --\n", Rp,
            I, Uc, T0, circuit.Params.Rk)
    }

    circuit.DrawPlot(tRes, IRes, "I(t)", "t", "I", "data/it.png")
    circuit.DrawPlot(tRes, UcRes, "U(t)", "t", "U", "data/ut.png")
    circuit.DrawPlot(tRes, RpRes, "Rp(t)", "t", "Rp", "data/rpt.png")
    circuit.DrawPlot(tRes, T0Res, "T0(t)", "t", "T0", "data/t0t.png")
    circuit.DrawPlot(tRes, IRpRes, "I(t) * Rp(t)", "t", "I * Rp", "data/irpt.png")
}
```

Результат работы программы

На рисунках 1 – 5 представленны графики зависимости от времени импульса t: $I(t)$, $U(t)$, $R_p(t)$, $I(t) * R_p(t)$, $T_0(t)$ соответственно при исходных данных. Интервал: [0, 0.0008], шаг $h = 10^{-6}$.

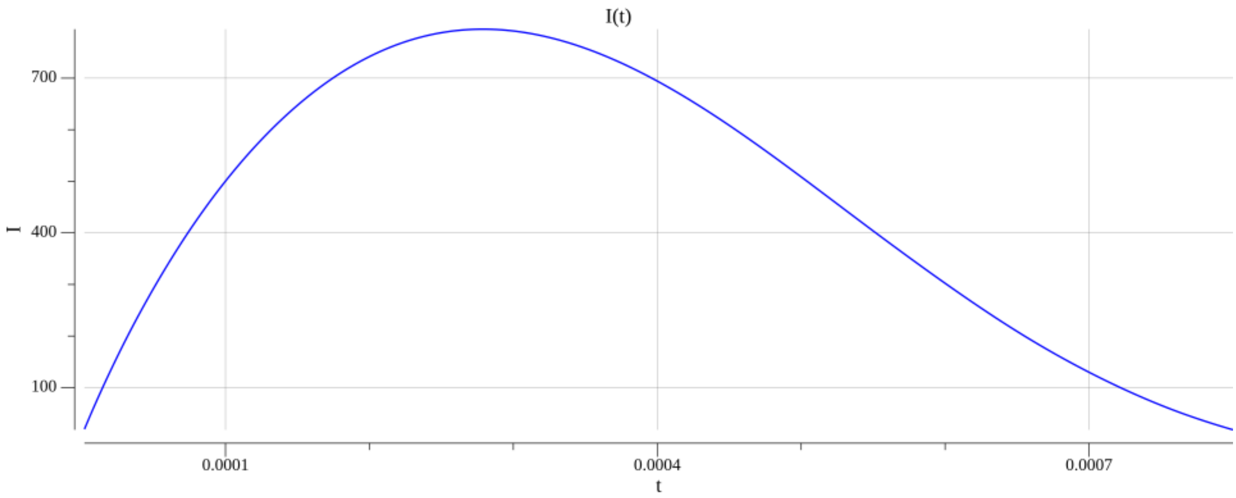

Рисунок 1 – График зависимости $I(t)$

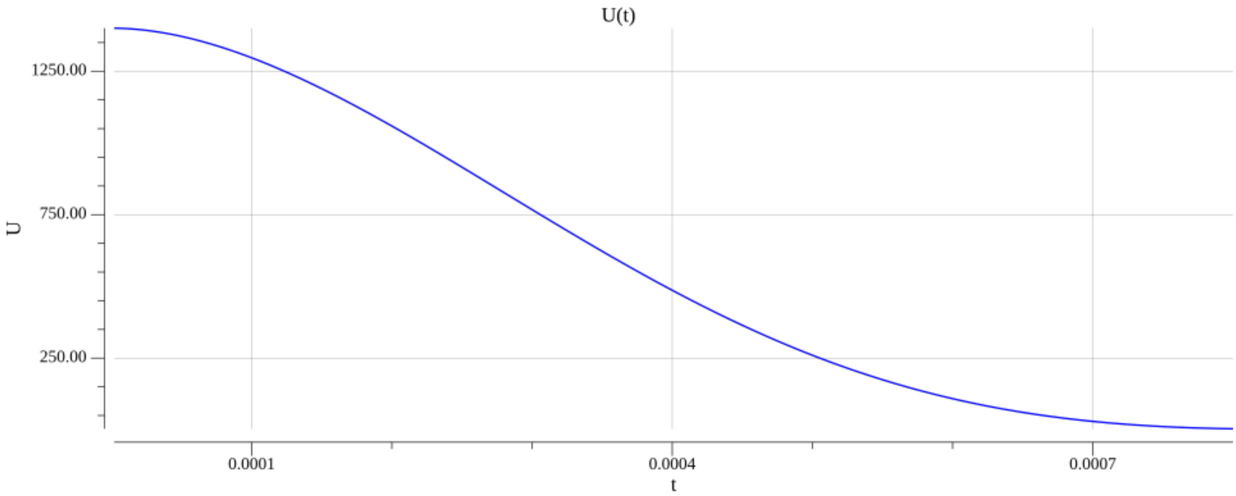

Рисунок 2 – График зависимости $U(t)$

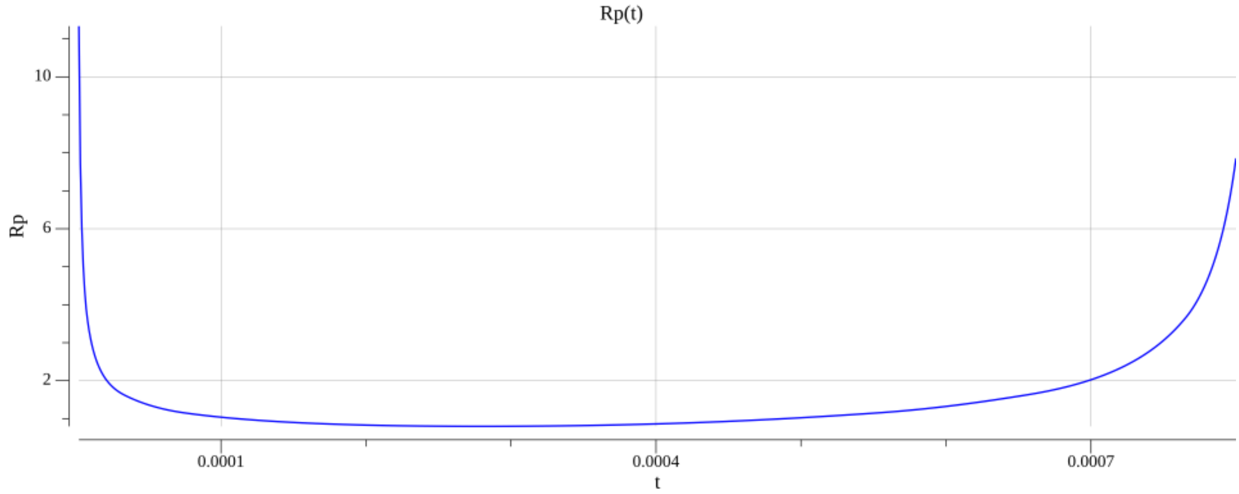

Рисунок 3 – График зависимости $R_p(t)$

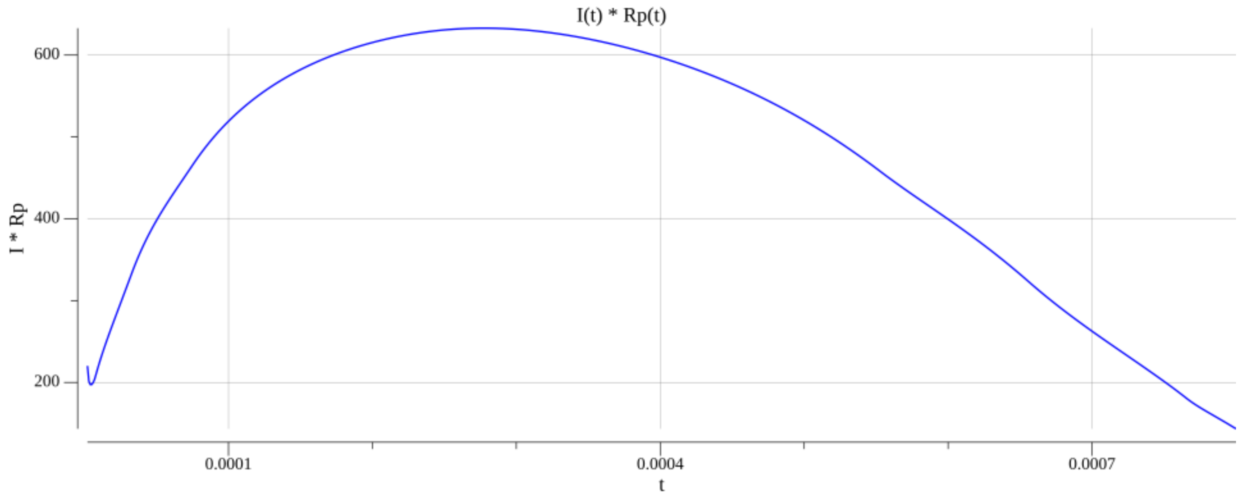

Рисунок 4 – График зависимости $I(t) \cdot R_p(t)$

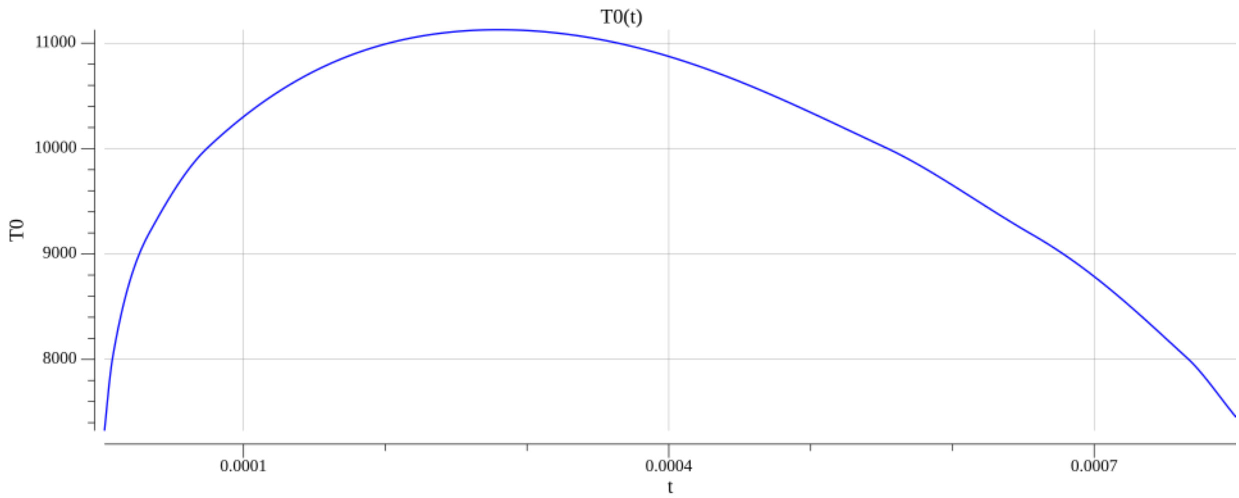

Рисунок 5 – График зависимости $T_0(t)$

На рисунке 6 представлен график $I(t)$, при $R_k + R_p = 0$. Интервал: [0, 0.008], шаг $h = 10^{-6}$.

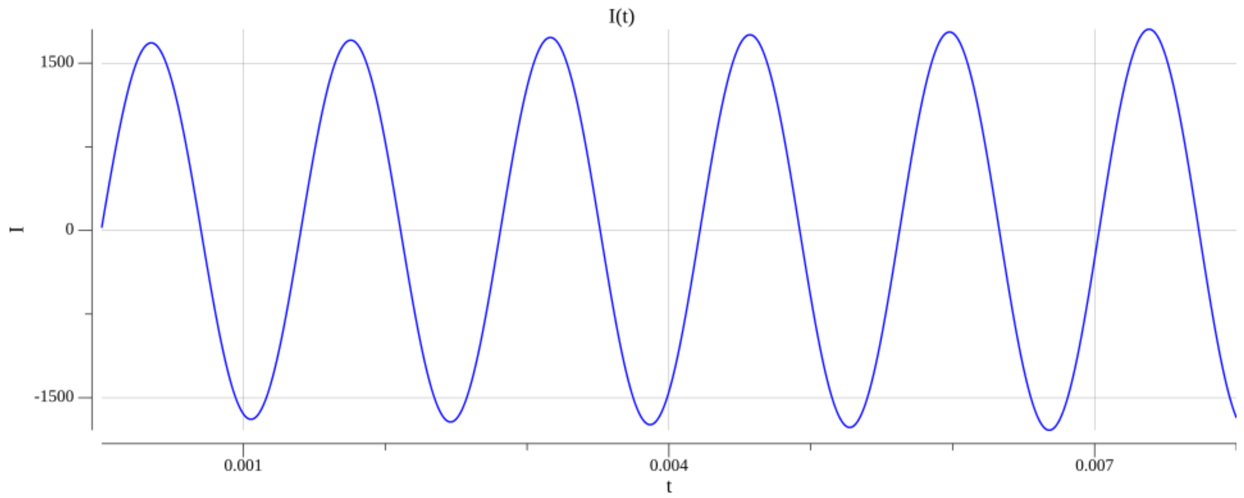

Рисунок 6 – График зависимости $I(t)$ при $R_k + R_p = 0$

На рисунке 7 представлен график $I(t)$, при $R_k + R_p = 200$. Интервал: [0, 0.00002], шаг $h = 10^{-7}$.

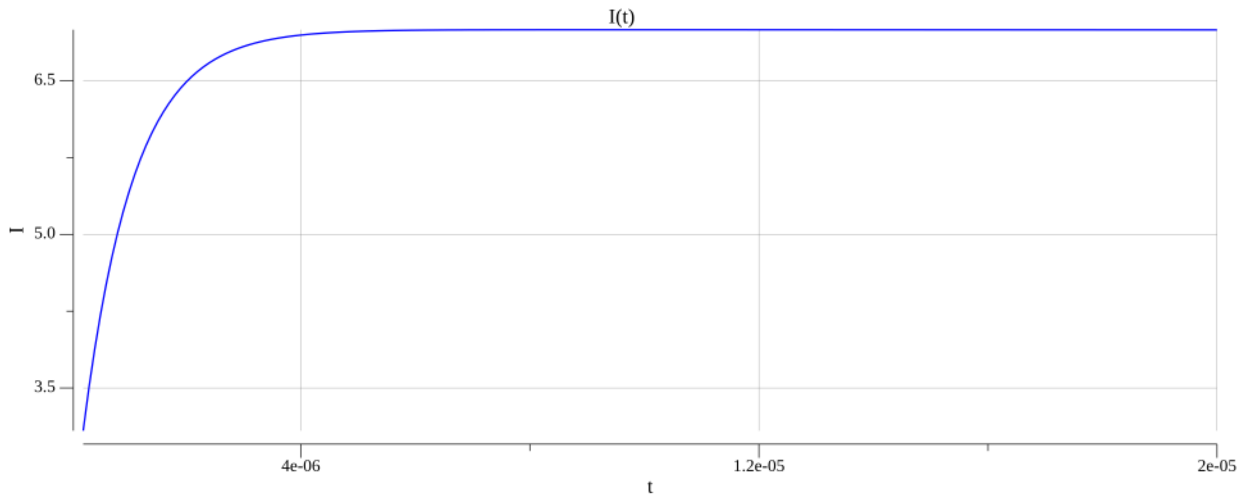

Рисунок 7 – График зависимости $I(t)$ при $R_k + R_p = 200$

На рисунках 8 – 13 представлены результаты исследования влияния параметров контура C_k, L_k, R_k на длительность импульса t.

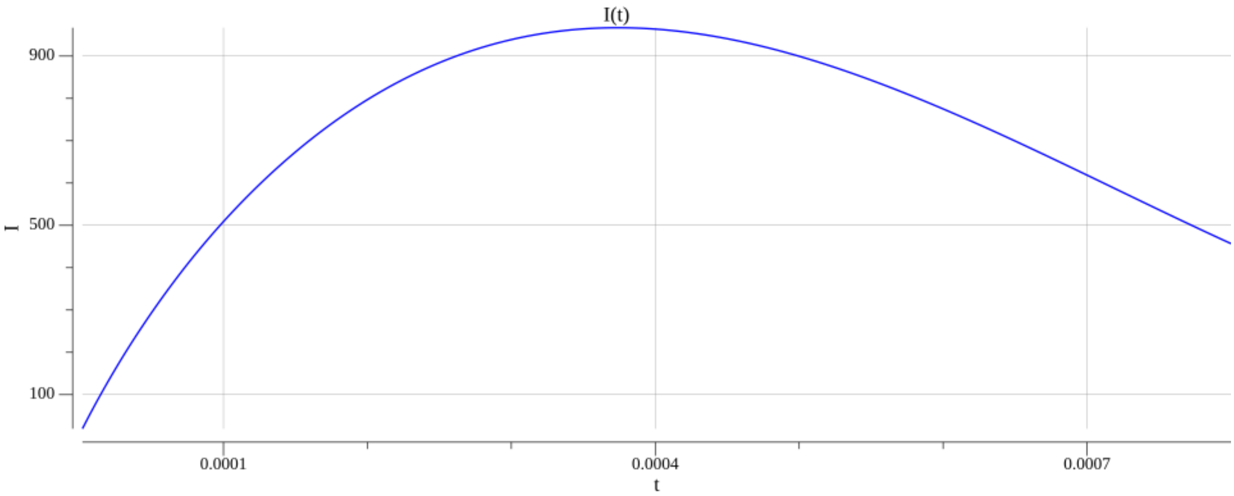

Рисунок 8 – График зависимости $I(t)$ при увеличении начального значения C_k в 2 раза

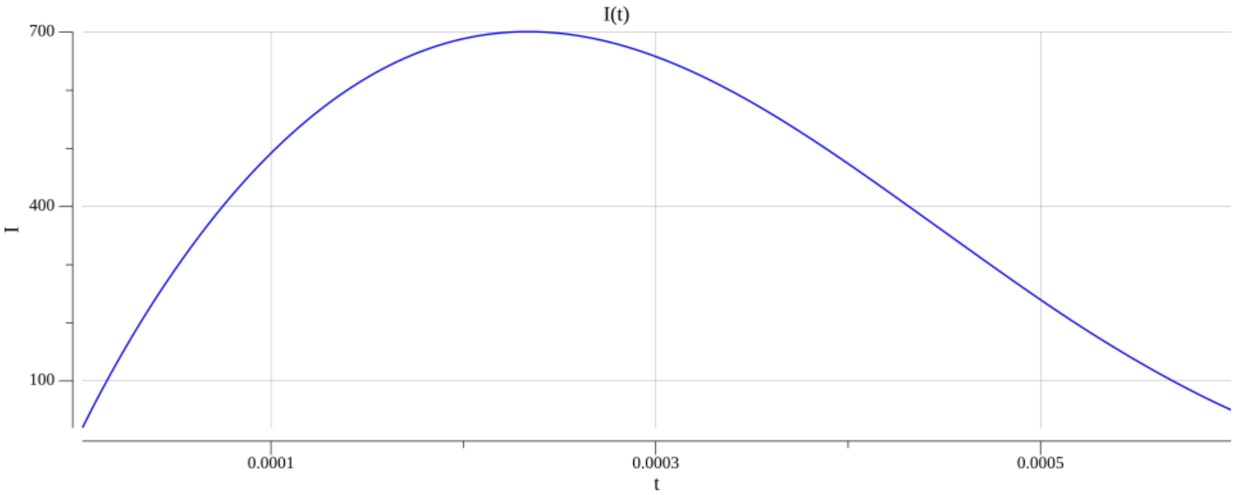

Рисунок 9 – График зависимости $I(t)$ при уменьшении начального значения C_k в 1.5 раза

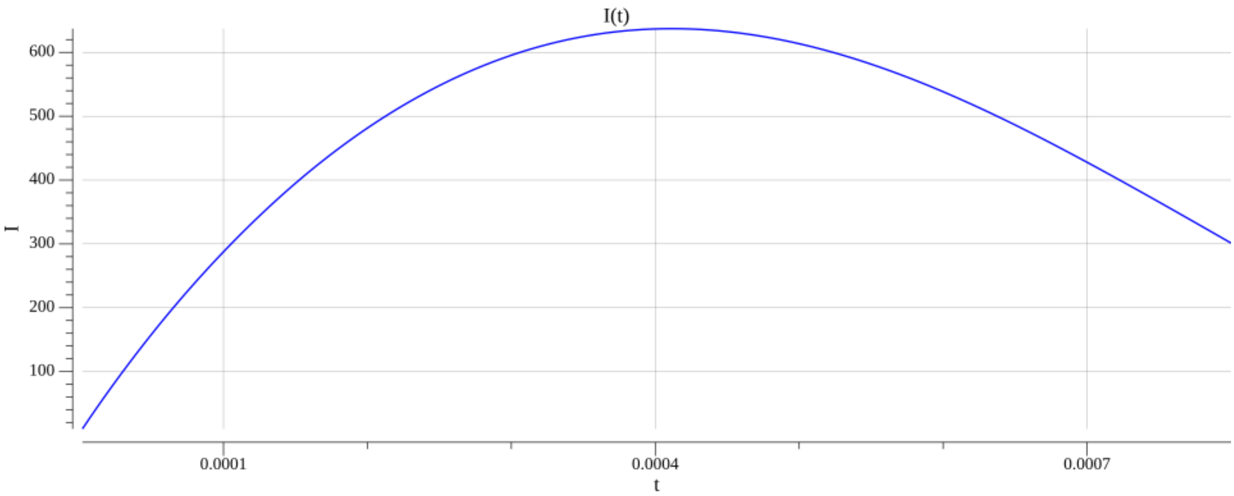

Рисунок 10 – График зависимости $I(t)$ при увеличении начального значения L_k в 2 раза

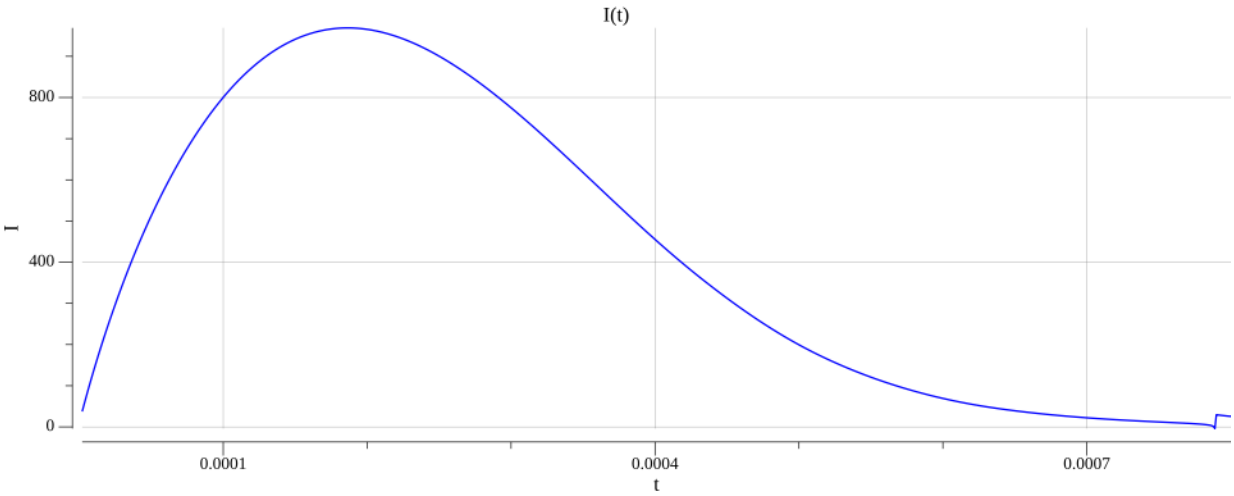

Рисунок 11 – График зависимости $I(t)$ при уменьшении начального значения L_k в 2 раза

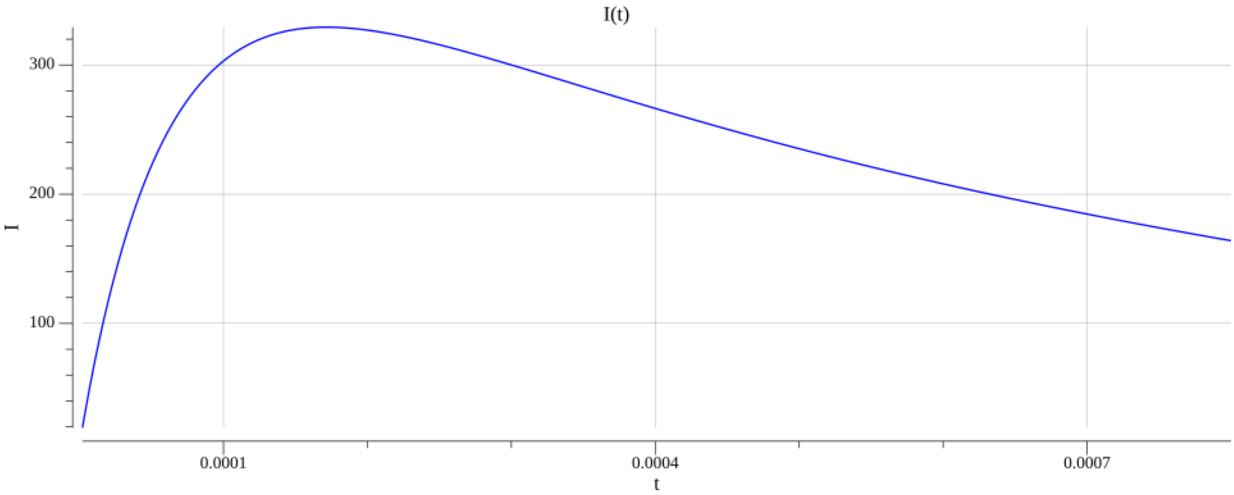

Рисунок 12 – График зависимости $I(t)$ при увеличении начального значения R_k в 10 раз

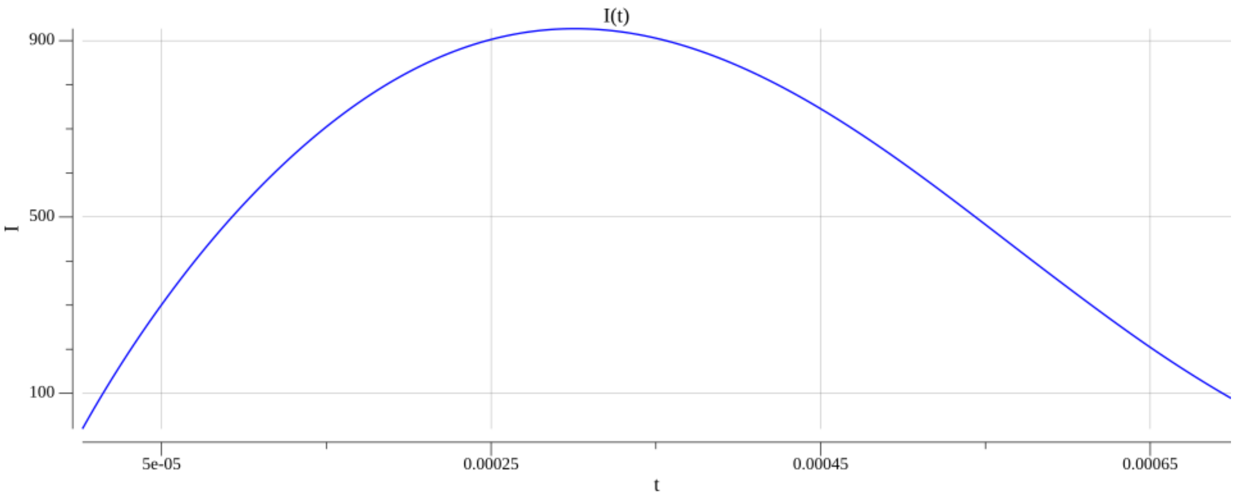

Рисунок 13 – График зависимости $I(t)$ при уменьшении начального значения R_k в 10 раз

В результате исследования было выявлено, что:

- увеличение C_k приводит к увеличению длительности импульса t;
- уменьшение C_k приводит к уменьшению длительности импульса t;
- увеличение L_k приводит к увеличению длительности импульса t;
- уменьшение L_k приводит к уменьшению длительности импульса t;
- увеличение R_k приводит к увеличению длительности импульса t;
- уменьшение R_k приводит к уменьшению длительности импульса t;

Ответы на контрольные вопросы

Вопрос 1. Какие способы тестирования программы, кроме указанного в п. 2, можете предложить ещё?

Ответ. В качестве еще одного способа тестирования можно из цепи убрать лампу: при небольших значениях R_k получатся затухающие колебания, при больших значения R_k получится апериодическое затухание.

Вопрос 2. Получите систему разностных уравнений для решения сформулированной задачи неявным методом трапеций. Опишите алгоритм реализации полученных уравнений.

Ответ.

$$U_{n+1} = U_n + \frac{h}{2} + f(x_n, u_n) + f(x_{n+1}, u_{n+1}) + O(h^2) \tag{1}$$

$$\begin{cases} \frac{dI}{dT} = \frac{U-(R_k+R_p(I))I}{L_k} \\ \frac{dU}{dt} = -\frac{I}{C_k} \end{cases} \tag{2}$$

$$I_{n+1} = I_n + \frac{h}{2}(\frac{U_n - (R_k + R_p(I_n))I_n}{L_k} + \frac{U_{n+1} - (R_k + R_p(I_{n+1}))I_{n+1}}{L_k}) \tag{3}$$

$$U_{n+1} = U_n + \frac{h}{2}(-\frac{I_n}{C_k} - \frac{I_{n+1}}{C_k}) = U_n - \frac{h}{2}(\frac{I_n + I_{n+1}}{C_k}) \tag{4}$$

Подставляя (4) в (3), имеем:

$$I_{n+1} = I_n + \frac{h}{2L_k}(2U_n - (R_k + R_p(I_n) + \frac{h}{2C_k})I_n - (R_k + R_p(I_{n+1}) + \frac{h}{2C_k})I_{n+1}) \tag{5}$$

Вопрос 3. Из каких соображений проводится выбор численного метода того или иного порядка точности, учитывая, что чем выше порядок точности метода, тем он более сложени требует, как правило, больших ресурсов вычислительной системы?

Ответ. При выборе численного метода оценивается погрешность для частного случая вида правой части дифференциального уравнения: $\varphi(x,\nu) = \varphi(x)$. Если $\varphi(x,\nu)$ непрерывна и ограничена и ограничены и непрерывны её четвертые производные, то наилучший результат достигается при $y_{n+1} = y_n + \frac{k_1+2k_2+2k_3+k_4}{6}$, где

$k_1 = h_n f(x_n, y_n)$

$k_2 = h_n f(x_n + \frac{h_n)}{2}, y_n + \frac{k_1}{2}$

$k_3 = h_n f(x_n + \frac{h_n)}{2}, y_n + \frac{k_2}{2}$

$k_4 = h_n f(x_n + \frac{h_n)}{2}, y_n + k_3$

Если $\varphi(x,\nu)$ не имеет таких производных, то четвертый порядок схемы не может быть достигнут и стоит применять более простые схемы.